

# Linguaggi e tecnologie per il Web

JavaScript

Ing. Massimo Nannini

# Introduzione

- **JavaScript** è un linguaggio di scripting **lato client** utilizzato per rendere dinamico il codice HTML.
- Il documento HTML è generato staticamente.
- Il codice JavaScript è immerso nel documento HTML ma viene **eseguito dinamicamente** solo al momento della richiesta del web client (browser).
- Principali usi:
  - posizionamento dinamico degli oggetti;
  - validazione campi dei moduli;
  - effetti grafici;

# Introduzione

- JavaScript è un linguaggio di programmazione di tipo script:
  - Non è compilato
  - E' ad alto livello
- La sua caratteristica principale è di essere nato espressamente per il WWW (è stato introdotto da Netscape nel 1995) e di essere supportato dalla maggior parte dei browser in uso.

# Introduzione

- Non può essere adoperato per costruire programmi complessi o con particolari requisiti di prestazioni, tuttavia può essere impiegato per implementare velocemente piccole procedure e controlli.
- Il suo campo d'impiego naturale è la gestione sul web client (browser) di alcuni elementi delle applicazioni per [WWW](http://www.gemaxconsulting.it).

# PRIMA PARTE

## Il linguaggio JavaScript

# Sintassi

- Un programma JavaScript è composto da una sequenza di istruzioni terminate dal punto e virgola (;)
- Un insieme di istruzioni delimitate da parentesi graffe ( { e } ) costituiscono un blocco.

# Sintassi

- Un **identificatore** è una sequenza di simboli che identifica un elemento all'interno del programma
- Un identificatore può essere composto da lettere e numeri, ma:
  - non deve contenere elementi di punteggiatura o spazi, e
  - deve cominciare con una lettera.

# Sintassi

- Questi sono identificatori validi
  - nome
  - Codice\_Fiscale
  - a0
- Questi sono identificatori non validi
  - 0a
  - x.y
  - Partita IVA



# Sintassi

- Il linguaggio è **case-sensitive**, ovvero distingue le lettere maiuscole dalle minuscole
- I commenti sono delimitati da `/ *` e `* /`
- Il simbolo `//` indica che il testo seguente sulla medesima riga è un commento

# Variabili

- Una variabile è un'area di memoria contenente informazioni che possono “variare” nel corso dell'esecuzione del programma.
- Una variabile è caratterizzata da un nome identificativo e dal tipo dell'informazione contenuta.

# Definizione di variabili

- Prima di essere adoperata una variabile deve essere definita

```
var eta;
```

- La definizione stabilisce il nome della variabile, mentre il tipo dipende dall'assegnazione

```
eta = 35;    //intero
```

```
nome = "Mario Rossi"; //stringa
```

# Definizione di variabili

- Il tipo di una variabile dipende dall'ultima assegnazione, quindi una variabile può cambiare tipo nel corso del suo ciclo di vita

```
x = 10; //intero
```

```
...
```

```
x = "a"; //stringa
```

- JavaScript è un linguaggio “debolmente tipato”

# Tipi dato predefiniti

- Number
- Boolean
- Null
- String
- Date
- Array

# Tipo Number

- Una variabile di tipo Number assume valori numerici interi o decimali

```
var x = 10;      //valore intero
```

```
var y = -5.3;    //valore decimale
```

- Sono definite le operazioni aritmetiche fondamentali ed una serie di funzioni matematiche di base

# Tipo Boolean

- Una variabile di tipo Boolean assume i soli valori della logica booleana vero e falso

```
var pagato = true; //valore logico vero  
var consegnato = false; //valore logico  
falso
```

- Sono definite le operazioni logiche fondamentali (AND, OR e NOT)

# Tipo Boolean

AND:

| X     | Y     | X AND Y |
|-------|-------|---------|
| true  | true  | true    |
| true  | false | false   |
| false | true  | false   |
| false | false | false   |



# Tipo Boolean

OR:

| X     | Y     | X OR Y |
|-------|-------|--------|
| true  | true  | true   |
| true  | false | true   |
| false | true  | true   |
| false | false | false  |

# Tipo Boolean

NOT:

| X     | NOT X |
|-------|-------|
| true  | false |
| false | true  |

# Tipo Null

- Si tratta di un tipo che può assumere un unico valore

```
var a = null;
```

- Serve ad indicare che il contenuto della variabile è non significativo

```
var sesso = "f";
```

```
var militesente = null;
```

# Tipo String

- Una variabile di tipo String contiene una sequenza arbitraria di caratteri
- Un valore di tipo Stringa è delimitato da apici ( ' ' ) o doppi-apici ( " " )

```
var nome = "Mario Rossi";  
var empty = ""; //Stringa vuota  
var empty2 = new String(); //Stringa vuota  
var str = 'Anche questa è una stringa';  
var str2 = new String("Un'altra stringa");
```

# Tipo Date

- Una variabile di tipo Date rappresenta un istante temporale (data ed ora)

```
var adesso = new Date();  
var natale2012 = new Date(2012, 11, 25);  
var capodanno2013 = new Date("Gen 1  
2013");
```

- E' definito l'operatore di sottrazione (-) tra due date che restituisce la differenza con segno espressa in millisecondi

# Tipo Array

- Un array è un vettore monodimensionale di elementi di tipo arbitrario

```
var v = new Array(); //Vettore vuoto  
var w = new Array("Qui", "Quo", "Qua");  
var u = new Array("Lun", "Mar", "Mer",  
    "Gio", "Ven", "Sab", "Dom");
```

- Non è necessario specificare la dimensione

# Assegnazione

- L'assegnazione è l'operazione fondamentale nei linguaggi di programmazione imperativa  
**id = expr**
- Dapprima viene “valutata” l'espressione **expr**, quindi il risultato viene “assegnato” alla variabile **id**

# Assegnazione

- Si consideri il seguente frammento di codice

```
var x = 10;
```

```
var y = 7;
```

```
var z = 3 * (x - y);
```

- Al termine dell'esecuzione la variabile  $z$  assume il valore 9



# Espressioni

- Un'espressione è composta da
  - identificatori di variabili  
`x, nome, eta, importo, ...`
  - costanti  
`10, 3.14, "<HTML>", ...`
  - operatori
  - parentesi ( ( e ) ) per alterare le regole di precedenza tra operatori

# Operatori aritmetici

- Operano tra valori (costanti o variabili) numerici
  - Somma ( $1 + 1$ )
  - Sottrazione ( $3 - 5$ )
  - Moltiplicazione ( $3 * 4$ )
  - Divisione ( $6 / 4$ )
  - Modulo ( $6 \% 5$ )
  - Cambiamento di segno ( $-3$ )

# Operatori di pre/post incremento/decremento

- Derivano dal linguaggio C (e sono presenti in C++ e Java)
- Alterano e restituiscono il valore di una variabile
- Consentono di scrivere codice compatto ed efficiente

# Operatori di pre/post incremento/decremento

```
var x = 10;
```

```
var y = x++; // y=10 e x=11
```

```
var z = ++x; // z=12 e x=12
```

```
var w = x--; // w=12 e x=11
```

```
var v = --x; // v=10 e x=10
```

# Operatori su stringhe

- L'unica operazione possibile in un'espressione è la concatenazione di stringhe

```
nomeCompleto = titolo + " " + nome + " "  
              + cognome
```

```
indirizzo = recapito + ", " + numCivico +  
              " " + CAP + " - " + citta + " (" + prov  
              + ") "
```

# Operatori su stringhe

- Esiste una forma più compatta per esprimere l'accodamento
- L'istruzione seguente

```
str = str + newStr;
```

è equivalente a

```
str += newStr;
```

# Operatori su vettori

- L'operatore definito sui vettori è l'accesso ad un determinato elemento dato l'indice

```
v[3] = 10;
```

```
a[i] = b[i] * c[i];
```

```
p = v[1];
```

- Il primo elemento di un vettore ha sempre l'indice 0 (come in C/C++ e Java)

# Operatori sui vettori

- Se si assegna un valore ad un elemento non presente questo viene creato

```
var v = new Array();  
v[0] = 1;
```

- Se si accede al valore di un elemento non presente si ottiene un valore indefinito

```
var v = new Array();  
var a = v[0];
```



# Operatori relazionali

- Confrontano due valori e restituiscono l'esito come valore booleano
  - Uguaglianza (`==`)
  - Disuguaglianza (`!=`)
  - Minore (`<`)
  - Maggiore (`>`)
  - Minore o uguale (`<=`)
  - Maggiore o uguale (`>=`)

# Operatori logici

- Operano tra valori (costanti o variabili) booleani
  - AND ( & & )
  - OR ( | | )
  - NOT ( ! )
- Solitamente gli operandi sono l'esito di un confronto

# Condizioni

- L'utilizzo di operatori relazionali e logici consente di formulare delle condizioni che possono essere utilizzate per controllare l'esecuzione del programma

```
(metodoPagamento=="contrassegno") &&  
(!residenteInItalia)
```

# Controllo dell'esecuzione

- L'esecuzione di un programma è generalmente sequenziale
- Tuttavia in determinate “condizioni” può essere necessario eseguire solo alcune istruzioni, ma non altre, oppure ripetere più volte un'operazione

# Istruzione if

- L'istruzione **instr** viene eseguita solo se la condizione **cond** risulta vera

```
if (cond)  
    instr
```

- L'istruzione **instr** può essere sostituita da un gruppo di istruzioni tra parentesi graffe ( { e } )

# Istruzione if

- Una costruzione alternativa prevede la presenza di una seconda istruzione da eseguire nel caso la condizione risulti falsa

```
if (cond)  
    instr_then  
else  
    instr_else
```

# Istruzione if

```
if (scelta=="NO") {  
    // Se la scelta è NO  
    ...  
} else {  
    // Altrimenti  
    ...  
}
```

# Istruzione for

- Viene dapprima eseguita l'istruzione **init**, quindi l'istruzione **instr** viene ripetuta finché la condizione **cond** risulta vera, dopo ogni ripetizione viene eseguita l'istruzione **next**

```
for (init; cond; next)  
    instr
```



# Istruzione for

- Inizializzare a 0 gli n elementi del vettore a

```
for (var i=0; i<n; i++)  
    a[i]=0;
```

- Copiare gli n elementi del vettore a nel vettore b

```
for (var i=0; i<n; i++)  
    b[i]=a[i];
```

# Istruzione while

- L'istruzione **instr** viene eseguita finché la condizione **cond** risulta essere verificata

```
while (cond)
```

```
    instr
```

- Un'istruzione for può essere espressa come

```
init;
```

```
while (cond) { instr; next; }
```

# Funzioni

- Una funzione è un elemento di un programma che calcola un valore che dipende “funzionalmente” da altri

$$y \leftarrow f(x)$$

$$y \leftarrow \log_{10}(x)$$

- L'utilizzo delle funzioni nella programmazione strutturata aumenta la modularità e favorisce il riutilizzo

# Definizione di funzioni

- In JavaScript è possibile definire una o più funzioni all'interno di un programma

```
function name(arg0, arg1, ..., argn-1) {  
    ...  
}
```

- La funzione definita è identificata da **name** e dipende dagli argomenti **arg**<sub>0</sub>, **arg**<sub>1</sub>, ..., **arg**<sub>n-1</sub>

# Definizione di funzioni

- Somma di due numeri

```
function somma(a, b) {  
    return a+b;  
}
```

- La funzione viene “invocata” all’interno di un’espressione

```
var s = somma(1, 2);
```

# Invocazione di funzioni

- Quando una funzione viene invocata gli argomenti sono inizializzati con i valori specificati
- Quindi si procede con l'esecuzione delle istruzioni costituenti la funzione
- L'istruzione `return` restituisce il valore calcolato al chiamante

# Invocazione di funzioni

- La chiamata

```
x = somma (1, 2)
```

inizializza gli argomenti a e b rispettivamente ai valori 1 e 2

- L'istruzione

```
return a+b;
```

valuta l'espressione e restituisce il risultato (3) che viene assegnato alla variabile x

# Variabili locali e globali

- All'interno di una funzione è possibile definire delle variabili “confinare” all'interno della funzione stessa
- Tali variabili, dette **locali**, sono create all'atto dell'invocazione della funzione e sono distrutte al termine dell'esecuzione
- Il loro valore non è accessibile dall'esterno della funzione
- Ogni argomento di una funzione è una variabile locale definita implicitamente



# Variabili locali e globali

- Le variabili definite all'esterno di una funzione sono denominate, invece, **globali**
- A differenza delle variabili locali, le variabili globali sono accessibili da qualsiasi punto del programma, anche dall'interno di una funzione, sempre che in quest'ultima non sia stata definita una variabile locale con lo stesso nome

# Variabili locali e globali

- Si consideri il seguente frammento di codice

```
function f(...) {  
    ...  
    var x = 1;  
    ...  
}  
var x = -1;  
...f(...) ;
```

- La variabile globale `x` continua a valere -1

# Procedure

- Una funzione che non restituisce valori viene detta **procedura** (in analogia con il Pascal)
- Una procedura agisce in genere su variabili globali (*side-effect*)

# Funzioni predefinite

- In JavaScript sono presenti alcune **funzioni predefinite**
  - `isNaN(v)` verifica se `v` non è un numero
  - `isFinite(v)` verifica se `v` è finito
  - `parseFloat(str)` converte `str` in un numero decimale
  - `parseInt(str)` converte `str` in un numero intero

# Oggetti

- Un oggetto è un elemento caratterizzato da uno stato rappresentato mediante proprietà e da un insieme di azioni (o metodi) che può eseguire
- Oggetti caratterizzati dagli stessi metodi e dalle stesse proprietà, ma non dallo stesso stato, sono detti della stessa classe

# Oggetti

- JavaScript è un linguaggio orientato agli oggetti, tuttavia non possiede il costrutto di classe
- Molti tipi di dato fondamentali sono, in effetti, degli oggetti (String, Date, Array,...)

# Proprietà e metodi

- Una proprietà di un oggetto è assimilabile ad una variabile

```
cliente.nome = "Carlo Bianchi";
```

```
x = ordine.aliquotaIVA;
```

- Un metodo, invece, è simile ad una funzione

```
tot = ordine.calcolaTotale();
```

# Proprietà e metodi

- Esistono due sintassi alternative per accedere alle proprietà degli oggetti

`oggetto.proprieta`

`oggetto["proprieta"]`

- La seconda è utile quando il nome della proprietà viene determinato durante l'esecuzione del programma



# Oggetti di tipo String

- Proprietà
  - `length` lunghezza della stringa
- Metodi
  - `charAt(pos)` carattere alla posizione `pos`
  - `substring(start, end)` sottostringa dalla posizione `start` alla posizione `end`
  - `toUpperCase()/toLowerCase()` converte la stringa in maiuscolo/minuscolo
  - `indexOf(str, pos)` posizione della prima occorrenza della string `str` cercata a partire dalla posizione `pos`

# Oggetti di tipo Array

- Proprietà
  - `length` lunghezza del vettore
- Metodi
  - `sort()` ordina gli elementi del vettore
  - `reverse()` inverte l'ordine degli elementi del vettore

# Oggetti di tipo Date

- Metodi
  - `getXXX()` restituisce il valore della caratteristica `XXX` della data (es. *`getFullYear()`*).
  - `setXXX(val)` imposta il valore della caratteristica `XXX` della data (es. *`setFullYear(2013,1,1)`*);
  - `toString()` restituisce la data come stringa formattata

# Oggetti di tipo Date

| <b>Nome caratteristica</b> | <b>Significato</b>     |
|----------------------------|------------------------|
| Date                       | Giorno del mese        |
| Day                        | Giorno della settimana |
| FullYear                   | Anno                   |
| Minutes                    | Minuti                 |
| Hours                      | Ore                    |
| Month                      | Mese                   |
| Seconds                    | Secondi                |
| Time                       | Tempo (hh:mm:ss)       |

# Oggetto Math

- Proprietà
  - E costante di Eulero
  - PI pi greco
- Metodi
  - `abs(val)` valore assoluto
  - `ceil(val)/floor(val)` troncamento
  - `exp(val)` esponenziale
  - `log(val)` logaritmo
  - `pow(base, exp)` elevamento a potenza
  - `sqrt(val)` radice quadrata

# SECONDA PARTE

## Uso di JavaScript per il World Wide Web

# Integrazione con i browser web

- La caratteristica principale di JavaScript è di essere “integrabile” all’interno delle pagine web
- In particolare consente di aggiungere una logica procedurale alle pagine rendendole “dinamiche”
- A differenza di altre tecnologie, JavaScript funziona completamente sul client

# Integrazione con i browser web

- I campi di impiego tradizionali sono
  - Validazione dell'input dell'utente e controllo dell'interazione
  - Effetti visivi di presentazione
- Con l'avvento di HTML5 i potenziali usi di JavaScript sono notevolmente aumentati
- JavaScript è supportato da tutti i browser più diffusi



# Dynamic HTML (DHTML)

- L'integrazione degli script all'interno di una pagina web avviene in due modi
  - Associando funzioni JavaScript agli eventi che si intende gestire
  - Accedendo dalle funzioni JavaScript alle proprietà degli oggetti che costituiscono la pagina

# Dynamic HTML (DHTML)

- Una pagina “dinamica” (DHTML) è una pagina HTML contenente codice JavaScript associato a determinati eventi che implementa specifiche funzionalità
- Non bisogna confondere una pagina DHTML con una pagina generata dinamicamente dal server

# Inserire codice in una pagina

- Javascript viene inserito all'interno delle pagine web tra gli statements

**<script ...> .... </script>**

Il tag script ha due importanti attributi:

- Language
- Type

# Inserire codice in una pagina

- **Language:** specifica che linguaggio di scripting è in uso. Nelle nuove versioni di HTML non viene più spesso indicato.
- **Type:** questo è l'attributo che viene adesso utilizzato per indicare il linguaggio di scripting in uso.

# Inserire codice in una pagina

```
<script language="javascript" type="text/javascript">  
    JavaScript code  
</script>
```

# Inserire codice in una pagina

Vediamo quindi come è possibile inserire codice JavaScript in una pagina HTML.

Esistono essenzialmente tre modi per farlo:

- inserire **codice inline**
- scrivere **blocchi di codice nella pagina**
- importare file con codice **JavaScript esterno**

# Codice inline

Vediamo quindi come è possibile inserire codice JavaScript in una pagina HTML.

Esistono essenzialmente tre modi per farlo:

- inserire **codice inline**
- scrivere **blocchi di codice nella pagina**
- importare file con codice **JavaScript esterno**

# Codice inline

Il primo approccio, l'inserimento di codice *inline*, consiste nell'inserire direttamente le istruzioni JavaScript nel codice di un elemento HTML, assegnandolo ad un attributo che rappresenta un evento.

Chiariamo il concetto con un esempio:

```
<input Type="button" onclick= "javascript:alert('ciao')" value="Click me"/>
```

```
<a href="javascript:alert('Ciao!')">Cliccami</a>
```



# Blocco di codice

L'approccio inline può risultare immediato perché mette direttamente in relazione il codice da eseguire con un elemento HTML. Risulta però scomodo quando il codice da eseguire è più complesso o abbiamo necessità di definire variabili e funzioni. In questi casi possiamo ricorrere al tag **<script>** per inserire blocchi di codice in una pagina HTML, come nel seguente esempio:

```
<script type="text/javascript">alert('Ciao!')</script>
```

# Blocco di codice

Per motivi di compatibilità con i vecchi browser il codice è incluso in un commento HTML `<!--...//-->`

```
<script type="text/javascript">  
<!--  
    alert('Ciao!')  
//-->  
</script>
```

# Blocco di codice in <head>

```
<html>
<head>
  <script type="text/javascript">
    <!--
      function sayHello() {
        alert("Hello World")
      }
    //-->
  </script>
</head>
<body>
Click here for the result
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

# Blocco di codice in <body>

```
<html>
<head>
</head>
<body>
  <script type="text/javascript">
    <!--
      document.write("Hello World")
    //-->
  </script>
  <p>This is web page body </p>
</body>
</html>
```

# Blocco di codice in <head> e <body>

```
<html>
<head>
  <script type="text/javascript">
    <!--
    function sayHello() {
      alert("Hello World")
    }
    //-->
  </script>
</head>
```

```
<body>
  <script type="text/javascript">
    <!--
    document.write("Hello World")
    //-->
  </script>
  <input type="button"
    onclick="sayHello()" value="Say
    Hello" />
</body>
</html>
```

# Blocco di codice

Possiamo inserire blocchi di codice (e i relativi tag `<script>`) nella sezione **`<head>`** o nella sezione **`<body>`** della pagina HTML.

**Nota:** *se il codice JavaScript interagisce con un elemento HTML, occorre assicurarsi che tale elemento sia già stato analizzato dal parser HTML: così il corrispondente oggetto sarà disponibile in memoria. Questo spiega il perché talvolta troviamo uno o più blocchi di codice JavaScript in fondo alla pagina prima della chiusura del tag `</body>`*

# Javascript esterno

Consiste nel collegare alla pagina HTML, codice JavaScript presente **in** un file esterno. Questa tecnica permette di agganciare script e librerie **in** modo detto non intrusivo, con il vantaggio di una separazione netta tra la struttura del documento e il codice, come accade per i fogli di stile CSS, che separano struttura e presentazione.

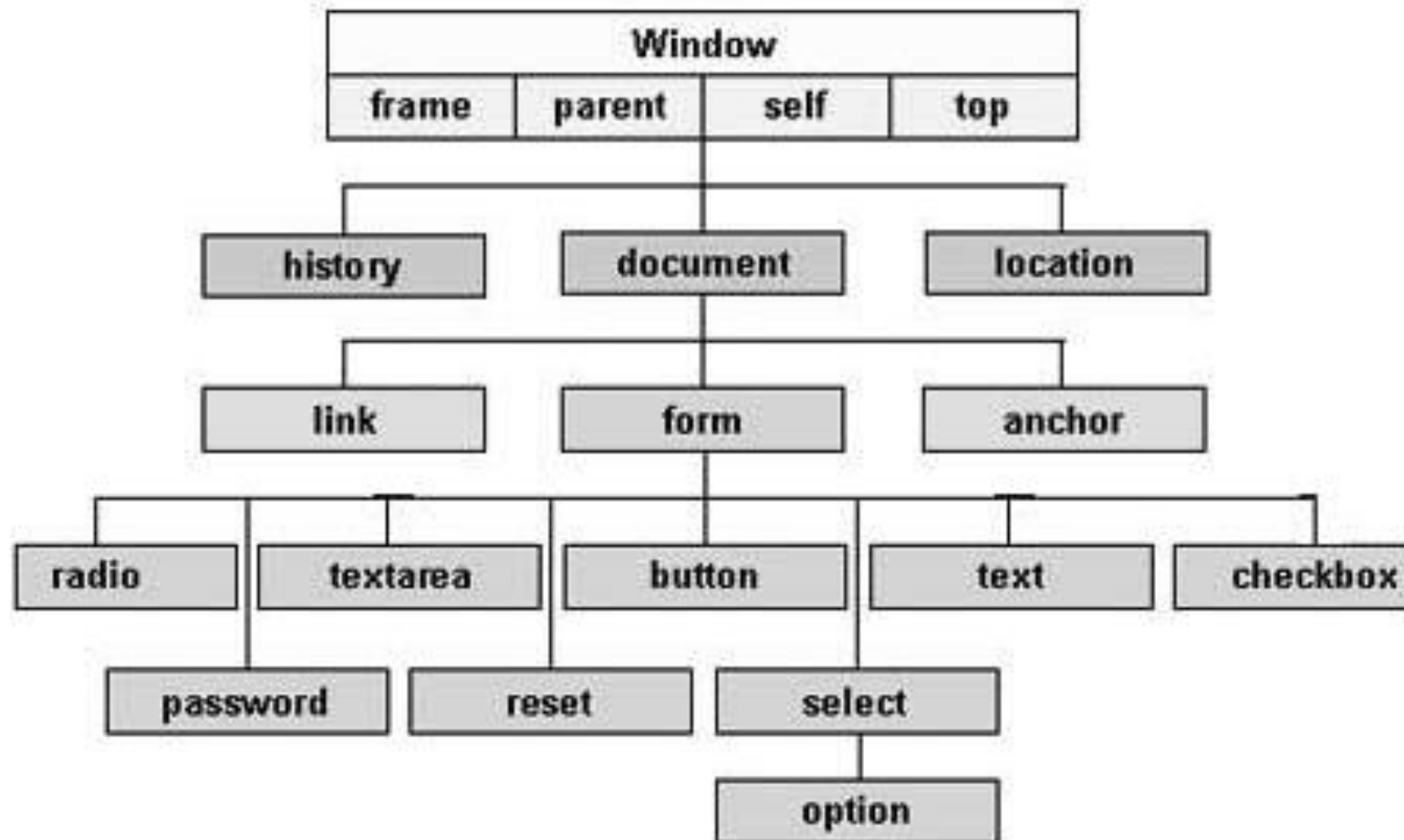
```
<script type="text/javascript" src="filename.js" ></script>
```

# Modello ad oggetti

- Un browser web esporta verso JavaScript un modello ad oggetti della pagina e dell’“ambiente” in cui la pagina è visualizzata
- Una funzione JavaScript adopera tali oggetti invocando i metodi e accedendo alle proprietà
- Il modello ad oggetti, a differenza del linguaggio, non è standard



# Modello ad oggetti



# Oggetto navigator

- L'oggetto `navigator` rappresenta l'istanza del browser in cui lo script è in esecuzione
- Proprietà
  - `appName` codice identificativo del browser
  - `appVersion` nome del browser
  - `userAgent` numero di versione

# Oggetto window

- Questo oggetto rappresenta la finestra in cui il documento corrente viene visualizzato
- Una funzione può accedere alle proprietà della finestra corrente, ma può creare e manipolare nuove finestre (pop-up)

# Oggetto window

- Proprietà
  - `title` titolo della finestra
  - `statusbar` testo mostrato sulla barra di stato
  - `location` URL del documento visualizzato
  - `outerHeight`, `outerWidth` dimensioni esterne
  - `innerHeight`, `innerWidth` dimensioni interne

# Oggetto window

- Modificare il titolo della finestra corrente

```
window.title = "Questo è il nuovo  
titolo";
```

- Accedere ad un nuovo documento

```
window.location =  
"http://www.yahoo.com/";
```

- Calcolare l'area in pixel della finestra

```
var area = window.innerWidth *  
window.innerHeight;
```

# Oggetto window

- Metodi
  - `open(location, title)` apre una nuova finestra
  - `alert(message)` visualizza il messaggio in una finestra di dialogo (utile per il debug)
  - `confirm(message)` visualizza il messaggio e richiede una conferma all'utente
  - `moveTo(x, y)` sposta la finestra alle coordinate indicate
  - `resizeTo(width, height)` dimensiona la finestra

# Oggetto window

- Creazione e posizionamento di una nuova finestra

```
var w = window.open("http://www.  
google.com/", "Google");  
w.moveTo(0, 0);
```

# Oggetto window

- **Visualizzazione di un messaggio**

```
window.alert("Attenzione si è  
verificato un errore");
```

- **Visualizzazione del browser in uso**

```
window.alert("Sei connesso con " +  
navigator.appName + " versione " +  
navigator.version);
```



# Oggetto window

- Richiesta conferma all'utente

```
if(confirm("Vuoi proseguire con  
l'operazione?")) {  
    //L'utente ha risposto SI  
    ...  
} else {  
    //L'utente ha risposto NO  
    ...  
}
```

# Oggetto history

- Rappresenta la sequenza di pagine visitate dall'utente
- Tale sequenza è rappresentata mediante un vettore
- Metodi
  - `back()` torna alla pagina precedente
  - `forward()` passa alla pagina successiva

# Oggetto document

- Rappresenta il documento HTML che costituisce la pagina visualizzata
- Non è possibile accedere a tutti gli elementi del documento
- Tuttavia è possibile accedere agli elementi dei moduli (form) ed alle proprietà di visualizzazione

# Oggetto document

- Inoltre, è possibile costruire on-the-fly il documento prima che questo sia stato completamente caricato e visualizzato

# Oggetto document

- Proprietà
  - `bgColor` colore dello sfondo
  - `fgColor` colore del testo
  - `forms` vettore dei moduli presenti nella pagina
  - `title` titolo del documento
  - `URL` indirizzo del documento
- Metodi
  - `write(string)` accoda `string` al documento, serve per la costruzione on-the-fly

# Oggetto document

- Supponendo che nel documento HTML sia definito un modulo di nome *modulo*

```
<FORM NAME="modulo"...>
```

...

```
</FORM>
```

# Oggetto document

- Si può accedere a tale oggetto in due diversi modi

```
document.forms["modulo"];
```

```
document.modulo;
```

- Ciò è possibile, in generale, per tutti gli elementi del documento con un attributo NAME

# Oggetto document

- Dal momento che la proprietà `forms` è di tipo `Array` è possibile accedervi anche tramite l'indice numerico dell'elemento

```
for (var i=0; i<document.forms.length; i++)  
{  
    //Accedi a document.forms[i]  
    ... = document.forms[i];  
    ...  
}
```



# Oggetto Form

- Un oggetto di questo tipo corrisponde ad un modulo all'interno di una pagina HTML
- Tramite le proprietà di questo oggetto è possibile accedere ai diversi elementi (o controlli) del modulo (inputbox, listbox, checkbox, ecc.)

# Oggetto Form

- Proprietà
  - `action` **valore dell'attributo** ACTION
  - `elements` **vettore contenente gli elementi del modulo**
  - `length` **numero di elementi del modulo**
  - `method` **valore dell'attributo** METHOD
  - `target` **valore dell'attributo** TARGET

# Oggetto Form

- Metodi
  - `reset()` azzera il modulo reimpostando i valori di default per i vari elementi
  - `submit()` invia il modulo

# Oggetto Form

- Supponendo che l'i-esimo elemento di un modulo `mod` sia denominato `nome_i` è possibile farvi riferimento in 3 modi diversi

```
document.mod.elements[i-1];
```

```
document.mod.elements["nome_i"];
```

```
document.mod.name_i;
```

- Attenzione l'indice del primo elemento di un vettore è sempre 0 (quindi l'i-esimo elemento ha indice i-1)

# Elementi dei moduli

- All'interno di un modulo possono comparire diversi tipi di elementi, corrispondenti ai vari costrutti HTML
- Ogni tipo ha proprietà e metodi specifici, per una trattazione approfondita si rimanda alla guida di riferimento del modello ad oggetti implementato dal browser

# Elementi dei moduli

| HTML   | JavaScript |
|--|------------|
| <code>&lt;INPUT TYPE="text"&gt;</code>         | Text       |
| <code>&lt;TEXTAREA&gt;&lt;/TEXTAREA&gt;</code> | Textarea   |
| <code>&lt;SELECT&gt;&lt;/SELECT&gt;</code>     | Select     |
| <code>&lt;INPUT TYPE="checkbox"&gt;</code>     | Checkbox   |
| <code>&lt;INPUT TYPE="radio"&gt;</code>        | Radio      |
| <code>&lt;INPUT TYPE="button"&gt;</code>       | Button     |

# Elementi dei moduli

- Tutti i tipi di elementi possiedono le seguenti proprietà
  - `name` nome dell'elemento
  - `value` valore corrente dell'elemento
- Gli elementi di tipo `Input` possiedono la proprietà `defaultValue` che contiene il valore predefinito del campo (attributo `VALUE` del tag HTML)

# Elementi dei moduli

- Gli elementi di tipo `Radio` e `Checkbox` possiedono la proprietà `checked` che indica se l'elemento è stato selezionato
- Gli elementi di tipo `Select` possiedono la proprietà `selectedIndex`, che contiene l'indice dell'elemento selezionato nella lista, e la proprietà `options`, che contiene il vettore delle scelte dell'elenco



# Elementi dei moduli

- E' possibile modificare i valori contenuti negli elementi dei moduli
- Pertanto è possibile utilizzare questi elementi anche per fornire risultati all'utente
- Se un elemento ha scopi esclusivamente di rappresentazione può essere marcato come READONLY

# Esempio: modulo di iscrizione

- Il modulo deve raccogliere i dati su un utente che vuole sottoscrivere un certo servizio
- Per ogni utente deve richiedere
  - nominativo, età, sesso
  - se desidera ricevere informazioni commerciali
  - il servizio cui desidera iscriversi

# Esempio: modulo di iscrizione

- Il modulo deve suggerire un'età di 18 anni ed il consenso all'invio di informazioni commerciali
- I servizi disponibili sono denominati “Servizio 1”, “Servizio 2” e “Servizio 3”
- Il modulo deve suggerire la sottoscrizione al primo servizio

# Esempio: modulo di iscrizione

- Per ogni dato si determina il tipo di elemento da inserire nel modulo
  - nominativo ed età con elementi di tipo Text
  - sesso con un elementi di tipo Radio
  - infoComm con un elemento di tipo Checkbox
  - servizio con un elemento di tipo Select
- Si deve, inoltre, inserire il pulsante di invio del modulo

# Esempio: modulo di iscrizione

- Per gestire la presentazione si adopera una tabella HTML che presenta sulla prima colonna il nome del campo e nella seconda gli elementi corrispondenti

# Esempio: modulo di iscrizione

```
<HTML>
```

```
<BODY>
```

```
<FORM NAME="iscrizione" ACTION="" METHOD="POST">
```

```
<TABLE>
```

```
...
```

# Esempio: modulo di iscrizione

...

```
<!-- Nominativo -->
```

```
<TR>
```

```
  <TD>Nominativo</TD>
```

```
  <TD>
```

```
    <INPUT NAME="nominativo" TYPE="text" SIZE="40">
```

```
  </TD>
```

```
</TR>
```

...

# Esempio: modulo di iscrizione

```
...  
<!-- Eta' -->  
<TR>  
  <TD>Et&agrave;</TD>  
  <TD>  
    <INPUT NAME="eta" TYPE="text" SIZE="3" VALUE="18">  
  </TD>  
</TR>  
...
```



# Esempio: modulo di iscrizione

```
<!-- Sesso -->  
<TR>  
  <TD>Sesso</TD>  
  <TD>  
    <INPUT NAME="sesso" TYPE="radio" VALUE="M">M  
    <INPUT NAME="sesso" TYPE="radio" VALUE="F">F  
  </TD>  
</TR>
```

# Esempio: modulo di iscrizione

...

```
<!-- Inform. commerciali -->
```

```
<TR>
```

```
  <TD>Inform. commerciali</TD>
```

```
  <TD>
```

```
    <INPUT NAME="infoComm" TYPE="checkbox" CHECKED>
```

```
  </TD>
```

```
</TR>
```

...

# Esempio: modulo di iscrizione

```
<!-- Servizio -->
<TR>
  <TD>Servizio</TD>
  <TD>
    <SELECT NAME="servizio">
      <OPTION VALUE="s1" SELECTED>Servizio 1</OPTION>
      <OPTION VALUE="s2">Servizio 2</OPTION>
      <OPTION VALUE="s3">Servizio 3</OPTION>
    </SELECT>
  </TD>
</TR>
```

# Esempio: modulo di iscrizione

...

```
<!-- Invio del modulo -->
```

```
<TR>
```

```
  <TD>
```

```
    <INPUT TYPE="submit" VALUE="Invia">
```

```
  </TD>
```

```
</TR>
```

...

# Esempio: modulo di iscrizione

...

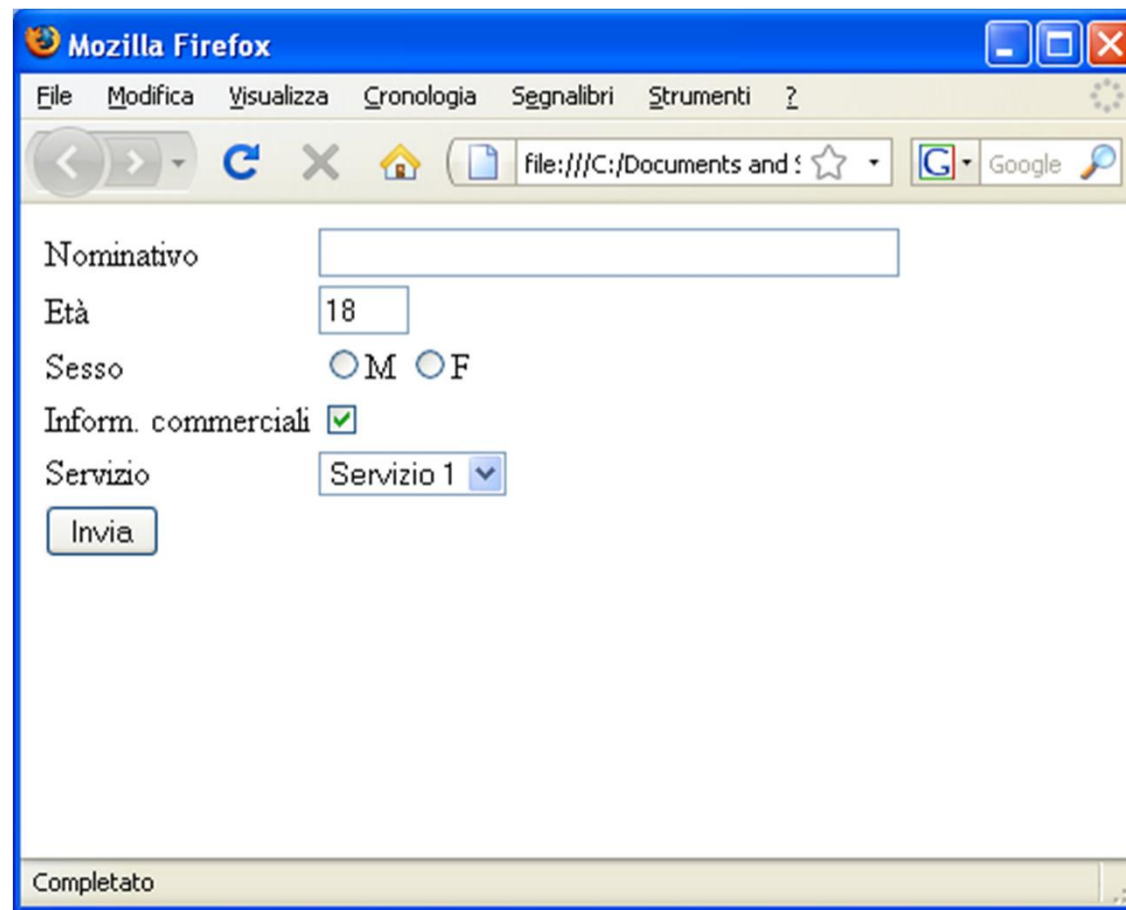
</TABLE>

</FORM>

</BODY>

</HTML>

# Esempio: modulo di iscrizione



The screenshot shows a Mozilla Firefox browser window with a registration form. The form fields are as follows:

| Field Label         | Value / Input Type                              |
|---------------------|---|
| Nominativo          | <input type="text"/>                            |
| Età                 | 18  |
| Sesso               | <input type="radio"/> M <input type="radio"/> F |
| Inform. commerciali | <input checked="" type="checkbox"/>             |
| Servizio            | Servizio 1 (dropdown menu)                      |

Below the form fields is an "Invia" button. At the bottom of the browser window, the status bar displays "Completato".

# Esempio: modulo di iscrizione

- Per accedere al nominativo immesso  
`document.iscrizione.nominativo.value`
- Per accedere all'età  
`document.iscrizione.eta.value`
- Per accedere al valore numerico dell'età  
`parseInt(document.iscrizione.eta.value)`

# Esempio: modulo di iscrizione

- Per visualizzare un messaggio relativo alla scelta di ricevere informazioni commerciali:

```
if (document.iscrizione.infoComm.checked)
    alert("Vuoi ricevere informazioni
        commerciali");
else
    alert("Non vuoi ricevere informazioni
        commerciali");
```



# Accedere ai campi per Id

Altro modo alternativo per accedere agli elementi è l'utilizzo della proprietà id.

```
<form name="contactForm">  
    <input type="text" name="nameBox" id="nameBoxId" />  
</form>
```

```
var userNameBox = document.getElementById("nameBoxId");  
var username = userNameBox.value;
```

Con questo metodo si riesce a trovare rapidamente un elemento corrispondente all'id senza dover attraversare tutto il percorso di tale elemento.

# Accedere ai campi per Id

```
<html>
<body>

<h1>What Can JavaScript Do?</h1>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button"
        onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'"> Click Me!</button>

</body>
</html>
```

# Eventi

- Ogni oggetto di un documento HTML “genera” degli **eventi** in risposta alle azioni dell’utente
- Ad esempio, l’evento `click` corrisponde al click del puntatore sull’oggetto
- Per gestire l’interazione con l’utente si associano funzioni JavaScript a particolari eventi

# Eventi

- Gli eventi generati da un oggetto dipendono dal tipo di quest'ultimo
- Oggetti `Form`
  - `onSubmit` invio del modulo
  - `onReset` azzeramento del modulo
- Oggetti `Button`
  - `onClick` click del puntatore

# Eventi

- **Oggetti Select, Text e Textarea**
  - onChange modifica del contenuto
  - onFocus selezione dell'elemento
- **Oggetti Radio e Checkbox**
  - onClick click del puntatore
  - onFocus selezione dell'elemento

# Intercettazione eventi

- Per intercettare l'evento  $E$  di un tag  $T$  ed associare l'esecuzione di una funzione  $f()$   
`<T onE="return f();">`
- Se il risultato della valutazione della funzione è `false` viene interrotta l'esecuzione del comando corrente, ad esempio l'invio di un modulo

# onClick Event Type

Questo evento è spesso utilizzato quando un utente clicca il tasto sinistro del proprio mouse.

```
<html>
<head>
  <script type="text/javascript">
    <!--
    function sayHello() {
      document.write ("Hello World")
    }
    //-->
  </script>
</head>
  <body>
    <p> Click the following button and see
    result</p>
    <input type="button" onclick="sayHello()"
    value="Say Hello" />
  </body>
</html>
```

# onSubmit Event Type

Questo evento è utilizzato quando si tenta di inviare un form. Si intercetta per inserire una validazione.

```
<html>
<head>
  <script type="text/javascript">
    <!--
    function validation() {
    all validation goes here
    .....
    return either true or false
    }
    //-->
  </script>

  <body>
    <form method="POST" action="t.cgi"
      onsubmit="return validate()">
      .....
      <input type="submit" value="Submit"
      />
    </form>
  </body>
</html>
```



# onMouseOver / onMouseOut

Questi eventi sono generati quando l'utente porta il mouse sopra un elemento o quando lo abbandona.

```
<html>
<head>
<script type="text/javascript">
  <!--
  function over() {
    document.write ("Mouse Over");
  }
  function out() {
    document.write ("Mouse Out");
  }
  //-->
</script>
</head>

<body>
  <p>Bring your mouse inside the
  division to see the result:</p>
  <div onmouseover="over()"
  onmouseout="out()">
    <h2> This is inside the division
  </h2>
  </div>
</body>
</html>
```

# HTML5 Standard Events

| Attribute        | Value  | Description  |
|------------------|--------|--|
| Offline          | Script | Triggers when the document goes offline                                      |
| Onabort          | Script | Triggers on an abort event   |
| onafterprint     | Script | Triggers after the document is printed                                       |
| onbeforeonload   | Script | Triggers before the document loads   |
| onbeforeonload   | Script | Triggers before the document is printed                                      |
| onblur           | Script | Triggers when the window loses focus   |
| oncanplay        | Script | Triggers when media can start play, but might has to stop for buffering      |
| oncanplaythrough | Script | Triggers when media can be played to the end, without stopping for buffering |
| onchange         | Script | Triggers when an element changes   |
| onclick          | Script | Triggers on a mouse click  |
| oncontextmenu    | Script | Triggers when a context menu is triggered                                    |
| ondblclick       | Script | Triggers on a mouse double-click   |
| ondrag           | Script | Triggers when an element is dragged  |

# HTML5 Standard Events

| Attribute        | Value  | Description  |
|------------------|--------|--|
| ondragend        | Script | Triggers at the end of a drag operation                            |
| ondragenter      | Script | Triggers when an element has been dragged to a valid drop target   |
| ondragleave      | Script | Triggers when an element leaves a valid drop target                |
| ondragover       | Script | Triggers when an element is being dragged over a valid drop target |
| ondragstart      | Script | Triggers at the start of a drag operation                          |
| ondrop           | Script | Triggers when dragged element is being dropped                     |
| ondurationchange | Script | Triggers when the length of the media is changed                   |
| onemptied        | Script | Triggers when a media resource element suddenly becomes empty      |
| onended          | Script | Triggers when media has reach the end                              |
| onerror          | Script | Triggers when an error occur                                       |
| onfocus          | Script | Triggers when the window gets focus                                |

# HTML5 Standard Events

| Attribute        | Value  | Description  |
|------------------|--------|--|
| onformchange     | Script | Triggers when a form changes   |
| onforminput      | Script | Triggers when a form gets user input   |
| onhaschange      | Script | Triggers when the document has change  |
| oninput          | Script | Triggers when an element gets user input                                     |
| oninvalid        | Script | Triggers when an element is invalid  |
| onkeydown        | Script | Triggers when a key is pressed   |
| onkeypress       | Script | Triggers when a key is pressed and released                                  |
| onkeyup          | Script | Triggers when a key is released  |
| onload           | Script | Triggers when the document loads   |
| onloadeddata     | Script | Triggers when media data is loaded   |
| onloadedmetadata | Script | Triggers when the duration and other media data of a media element is loaded |
| onloadstart      | Script | Triggers when the browser starts to load the media data                      |
| onmessage        | Script | Triggers when the message is triggered                                       |

# HTML5 Standard Events

| Attribute    | Value  | Description   |
|--------------|--------|---|
| onmousedown  | Script | Triggers when a mouse button is pressed                 |
| onmousemove  | Script | Triggers when the mouse pointer moves                   |
| onmouseout   | Script | Triggers when the mouse pointer moves out of an element |
| onmouseover  | Script | Triggers when the mouse pointer moves over an element   |
| onmouseup    | Script | Triggers when a mouse button is released                |
| onmousewheel | Script | Triggers when the mouse wheel is being rotated          |
| onoffline    | Script | Triggers when the document goes offline                 |
| onoinc       | Script | Triggers when the document comes online                 |
| ononline     | Script | Triggers when the document comes online                 |
| onpagehide   | Script | Triggers when the window is hidden                      |
| onpageshow   | Script | Triggers when the window becomes visible                |
| onpause      | Script | Triggers when media data is paused                      |

# HTML5 Standard Events

| Attribute          | Value  | Description  |
|--------------------|--------|--|
| onplay             | Script | Triggers when media data is going to start playing   |
| onplaying          | Script | Triggers when media data has start playing   |
| onpopstate         | Script | Triggers when the window's history changes   |
| onprogress         | Script | Triggers when the browser is fetching the media data   |
| onratechange       | Script | Triggers when the media data's playing rate has changed  |
| onreadystatechange | Script | Triggers when the ready-state changes  |
| onredo             | Script | Triggers when the document performs a redo   |
| onresize           | Script | Triggers when the window is resized  |
| onscroll           | Script | Triggers when an element's scrollbar is being scrolled   |
| onseeked           | Script | Triggers when a media element's seeking attribute is no longer true, and the seeking has ended |
| onseeking          | Script | Triggers when a media element's seeking attribute is true, and the seeking has begun           |

# HTML5 Standard Events

| Attribute      | Value  | Description  |
|----------------|--------|--|
| onselect       | Script | Triggers when an element is selected   |
| onstalled      | Script | Triggers when there is an error in fetching media data   |
| onstorage      | Script | Triggers when a document loads   |
| onsubmit       | Script | Triggers when a form is submitted  |
| onsuspend      | Script | Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched |
| ontimeupdate   | Script | Triggers when media changes its playing position   |
| onundo         | Script | Triggers when a document performs an undo  |
| onunload       | Script | Triggers when the user leaves the document   |
| onvolumechange | Script | Triggers when media changes the volume, also when volume is set to "mute"                                    |
| onwaiting      | Script | Triggers when media has stopped playing, but is expected to resume   |

# Intercettazione eventi

- Ad esempio, la funzione `valida()` verifica se i dati immessi nel modulo `modulo` sono corretti ed eventualmente procede con l'invio di quest'ultimo

```
<FORM NAME="modulo"  
  onSubmit="return valida();" ...>
```

...

```
</FORM>
```



# Validazione modulo di iscrizione

- Nel modulo di sottoscrizione del servizio è necessario indicare il nominativo del sottoscrittore
- Quindi il valore del campo corrispondente non deve essere una stringa vuota

# Validazione modulo di iscrizione

- Il modulo viene ridefinito come

```
<FORM NAME="iscrizione" ACTION=""  
      METHOD="POST" onSubmit="return  
      validaIscrizione();" >
```

- Nell'intestazione del documento viene definita una funzione `validaIscrizione()`

# Validazione modulo di iscrizione

```
function validaIscrizione() {  
    if (document.iscrizione.nominativo.value=="")  
    {  
        alert("Nominativo obbligatorio.  
        Impossibile procedere.");  
        return false;  
    }  
    ... //Altri controlli  
    return true;  
}
```

# Validazione form - Esempio

```
<html>
<head>
<title>Form Validation</title>
<script type="text/javascript">
<!--
    // Form validation code will come here.
    //-->
</script>
</head>
<body>
    <form action="/cgi-bin/test.cgi" name="myForm"
    onsubmit="return(validate());">
    <table cellspacing="2" cellpadding="2" border="1">
    <tr>
    <td align="right">Name</td>
```

# Validazione form - Esempio

```
<td><input type="text" name="Name" /></td>
</tr>
<tr>
<td align="right">EMail</td>
<td><input type="text" name="EMail" /></td>
</tr>
<tr>
<td align="right">Zip Code</td>
<td><input type="text" name="Zip" /></td>
</tr>
<tr>
<td align="right">Country</td>
<td>
```

# Validazione form - Esempio

```
<select name="Country">
<option value="-1" selected>[choose yours]</option>
<option value="1">USA</option>
<option value="2">UK</option>
<option value="3">INDIA</option>
</select>
</td>
</tr>
<tr>
<td align="right"></td>
<td><input type="submit" value="Submit" /></td>
</tr>
</table>
</form>
</body>
</html>
```

# Script di validazione – Esempio base

```
<script type="text/javascript">
<!--
  // Form validation code will come here.
  function validate()
  {
    if( document.myForm.Name.value == "" )
    {
      alert( "Please provide your name!" );
      document.myForm.Name.focus() ;
      return false;
    }

    if( document.myForm.Email.value == "" )
    {
      alert( "Please provide your Email!" );
      document.myForm.Email.focus() ;
      return false;
    }
  }
}
```

# Script di validazione – Esempio base

```
if( document.myForm.Zip.value == "" ||
isNaN( document.myForm.Zip.value ) ||
document.myForm.Zip.value.length != 5 )
{
    alert( "Please provide a zip in the format #####." );
    document.myForm.Zip.focus() ;
    return false;
}
if( document.myForm.Country.value == "-1" )
{
    alert( "Please provide your country!" );
    return false;
}
return( true );
}
//-->
</script>
```



# Script di validazione – Esempio base

```
<script type="text/javascript">
<!--
    function validateEmail()
    {
        var emailID = document.myForm.EMail.value;
        atpos = emailID.indexOf("@");
        dotpos = emailID.lastIndexOf(".");
        if (atpos < 1 || ( dotpos - atpos < 2 ))
        {
            alert("Please enter correct email ID")
            document.myForm.EMail.focus() ;
            return false;
        }
        return( true );
    }
//-->
</script>
```

# Proprietà di visualizzazione

- Tra le proprietà degli elementi del modello ad oggetti del documento sono presenti alcune specifiche della modalità di presentazione all'utente degli elementi medesimi
- La possibilità di poter accedere e manipolare tali caratteristiche permette di utilizzare JavaScript per ottenere sofisticati effetti di presentazione visiva
- Purtroppo, queste proprietà sono specifiche dei singoli browser

# Proprietà di visualizzazione

- Le proprietà di visualizzazione sono connesse con l'uso dei CSS
- Infatti, alcune proprietà degli stili possono essere modificate dinamicamente da funzioni JavaScript
- Tra le proprietà più utili per la creazione di effetti visivi abbiamo la visualizzazione ed il posizionamento degli elementi

# I cookies

Web browser e Servers usano il protocollo HTTP che è un protocollo "**stateless**".

In certe situazioni però può essere utile mantenere alcune informazioni che devono essere utilizzate da più pagine dello stesso sito.

In molte situazioni usare i cookies è il modo più efficiente di memorizzare e tenere traccia delle preferenze degli utenti.

# I cookies – Come funzionano ?

Il server invia alcuni dati al browser sotto forma di cookies. Il browser può accettare i cookies. Se è così una stringa di dati viene memorizzata sul disco del client.

Quando il visitatore accede ad un'altra pagina, il browser invia lo stesso cookie al server il quale in questo modo riesce a capire che si tratta del medesimo cookie inviato in precedenza.

# I cookies – Come funzionano ?

Il cookies è un record di dati 5 campi a di lunghezza variabile:

- **Expires:** è la data in cui il cookie scade. Se non indicata scade quando il browser viene chiuso.
- **Domain:** è il nome del dominio del sito internet.
- **Path:** è il percorso della pagina che ha impostato il cookie.
- **Secure:** se questo campo contiene la parola «secure», il cookies potrà essere recuperato solo da un server sicuro.
- **Name=Value:** il cookie può essere recuperato solo come coppia chiave-valore.

# I cookies – Come funzionano ?

I cookie sono stati originariamente progettati per la programmazione CGI.

I dati contenuti in un cookie vengono trasmessi automaticamente tra il browser e il server web, così che gli script CGI sul server possano leggere e scrivere i valori dei cookie che vengono memorizzati sul client.

# I cookies – Come funzionano ?

Anche JavaScript può manipolare i cookie utilizzando la proprietà cookie dell'oggetto Document.

JavaScript in grado di leggere, creare, modificare e cancellare i cookie che si applicano alla pagina Web corrente.



# Memorizzare i cookies

Anche JavaScript può manipolare i cookie utilizzando la proprietà cookie dell'oggetto Document.

JavaScript in grado di leggere, creare, modificare e cancellare i cookie che si applicano alla pagina Web corrente.

```
document.cookie = "key1=value1;key2=value2;expires=date";
```

# Memorizzare i cookies

**Nota:** i valori dei cookie non possono includere il punto e virgola, virgole o spazi. Per questo motivo, si consiglia di utilizzare la funzione **encodeURIComponent()** per codificare il valore prima di riporla nel cookie.

Se si esegue questa operazione, si avrà anche utilizzare la funzione corrispondente **decodeURI()** quando si legge il valore del cookie.

# Memorizzare i cookies - Esempio

```
<html>
<head>
<script type="text/javascript">
  <!--
  function WriteCookie()
  {
    if( document.myform.customer.value == "" ){
      alert ("Enter some value!");
      return;
    }
    cookievalue= encodeURIComponent(document.myform.customer.value) + ";";
    document.cookie="name=" + cookievalue;
    document.write ("Setting Cookies : " + "name=" + cookievalue );
  }
  //-->
</script>
```

# Memorizzare i cookies - Esempio

```
</head>
<body>
<form name="myform" action="">
    Enter name: <input type="text" name="customer"/>
    <input type="button" value="Set Cookie" onclick="WriteCookie();" />
</form>
</body>
</html>
```

Ora sulla macchina cliente è memorizzato un cookie di nome **name**.

E' possibile memorizzare cookies multipli usando key=value separati da virgola.

# Leggere i cookies

La lettura di un cookie è altrettanto semplice quanto la scrittura, perché il valore dell'oggetto **document.cookie** è il cookie.

Quindi è possibile utilizzare questa stringa ogni volta che si desidera accedere al cookie.

La stringa **document.cookie** contiene un elenco di coppie *nome=valore* separate da punto e virgola, dove *nome* è il nome di un cookie e *valore* è il valore della stringa.

# Leggere i cookies - Esempio

```
<html>
<head>
<script type="text/javascript">
  <!--
  function ReadCookie()
  {
    var allcookies = document.cookie;
    document.write ("All Cookies : " + allcookies );
    // Get all the cookies pairs in an array
    cookiearray = allcookies.split(';');
    // Now take key value pair out of this array
    for(var i=0; i<cookiearray.length; i++){
      name = cookiearray[i].split('=')[0];
      value = decodeURI(cookiearray[i].split('=')[1]);
      document.write ("Key is : " + name + " and Value is : " + value);
    }
  }
  //-->
</script>
```

# Leggere i cookies - Esempio

```
</head>
<body>
  <form name="myform" action="">
    <p> click the following button and see the result:</p>
    <input type="button" value="Get Cookie" onclick="ReadCookie()"/>
  </form>
</body>
</html>
```

**Nota:**

**length** è un metodo di classe Array che restituisce la lunghezza di un array.

# I cookies - Data di scadenza

È possibile estendere la durata di un cookie al di là della sessione corrente del browser impostando una data di scadenza e salvare la data di scadenza entro il cookie. Questo può essere fatto impostando l'attributo '**expires**' per una data e un'ora.



# Data di scadenza - Esempio

```
<html>
<head>
<script type="text/javascript">
  <!--
  function WriteCookie()
  {
    var now = new Date();
    now.setMonth( now.getMonth() + 1 );
    cookievalue = encodeURIComponent(document.myform.customer.value) + ";";
    document.cookie="name=" + cookievalue;
    document.cookie = "expires=" + now.toUTCString() + ";";
    document.write ("Setting Cookies : " + "name=" + cookievalue );
  }
  //-->
</script>
</head>
```

# Cancellare un cookie

A volte si vuole eliminare un cookie in modo che i tentativi successivi di leggerlo ritornino nullo. Per fare questo, è sufficiente impostare la data di scadenza di un tempo nel passato.

# Cancellare un cookie - Esempio

```
<html>
<head>
<script type="text/javascript">
  <!--
  function WriteCookie()
  {
    var now = new Date();
    now.setMonth( now.getMonth() - 1 );
    cookievalue = encodeURIComponent(document.myform.customer.value) + ";";
    document.cookie="name=" + cookievalue;
    document.cookie = "expires=" + now.toUTCString() + ";";
    document.write("Setting Cookies : " + "name=" + cookievalue );
  }
  //-->
</script>
</head>
```

# Cancellare un cookie - Esempio

```
<body>  
<form name="formname" action="">  
    Enter name: <input type="text" name="customer"/>  
    <input type="button" value="Set Cookie" onclick="WriteCookie()"/>  
</form>  
</body>  
</html>
```

# Page Refresh

È possibile aggiornare una pagina web utilizzando il metodo **location.reload**. Questo codice può essere chiamato automaticamente su un evento o semplicemente quando l'utente fa clic su un link.

Se si desidera aggiornare una pagina Web utilizzando un clic del mouse, quindi è possibile utilizzare il seguente codice:

```
<a href="javascript:location.reload(true)">Refresh Page</a>
```

# Page Auto-Refresh

È inoltre possibile utilizzare JavaScript per aggiornare la pagina automaticamente dopo un determinato periodo di tempo.

**setTimeout()** è una funzione JavaScript che può essere utilizzata per eseguire un'altra funzione dopo un dato intervallo di tempo.

# Page Auto-Refresh

```
<html>
<head>
<script type="text/JavaScript">
    <!--
    function AutoRefresh( t ) {
        setTimeout("location.reload(true);", t);
    }
    // -->
</script>
</head>
<body onload="JavaScript:AutoRefresh(5000);">
    <p>This page will refresh every 5 seconds.</p>
</body>
</html>
```

# Page Redirect

Vi potrebbe essere capitato di incontrare una situazione in cui cliccando un URL per raggiungere una pagina X, si viene reindirizzati a un'altra pagina Y. Succede grazie alla pagina di reindirizzamento. Questo concetto è diverso dal Refresh della pagina.



# Page Redirect – Esempio 1

```
<html>
<head>
<script type="text/javascript">
    <!--
    function Redirect() {
    window.location="http://www.gemaxconsulting.it";
    }
    //-->
</script>
</head>
<body>
    <p>Click the following button, you will be redirected to home page.</p>
<form>
    <input type="button" value="Redirect Me" onclick="Redirect();" />
</form>
</body>
</html>
```

# Page Redirect – Esempio 2

È possibile visualizzare un messaggio appropriato ai visitatori del sito prima di reindirizzamento a una nuova pagina.

```
<html>
<head>
<script type="text/javascript">
<!--
    function Redirect() {
        window.location="http://www.gemaxconsulting.it";
    }
    document.write ("You will be redirected to our main page in 10 seconds!");
    setTimeout('Redirect()', 10000);
//-->
</script>
</head>
<body>
</body>
</html>
```

# Page Redirect – Esempio 3

L'esempio seguente mostra come reindirizzare i visitatori del sito su una pagina diversa in base alle loro browser.

```
<html>
<head>
<script type="text/javascript">
    <!--
    var browsername=navigator.appName;
    if( browsername == "Netscape" )
    {
        window.location="http://www.location.com/ns.htm";
    }
    else if ( browsername == "Microsoft Internet Explorer")
    {
        window.location="http://www.location.com/ie.htm";
    }
    else
    {
        window.location="http://www.location.com/other.htm";
    }
    //-->
</script>
</head>
<body>
</body>
</html>
```

# Dialog Box

JavaScript supporta tre importanti tipi di finestre di dialogo. Queste finestre di dialogo possono essere utilizzate per sollevare e avviso, o per avere conferma su ogni ingresso o di avere un tipo di input da parte degli utenti.

- **Alert Dialog Box:** utilizzato per dare un avviso
- **Confirmation Dialog Box:** utilizzato per chiedere consenso all'utente. Ha due pulsanti (OK, Annulla)
- **Prompt Dialog Box:** utilizzando quando si deve richiedere all'utente un input. Ha il pulsante OK.

# Alert Dialog Box

```
<html>
<head>
<script type="text/javascript">
    <!--
    function Warn() {
        alert ("This is a warning message!");
        document.write ("This is a warning message!");
    }
    //-->
</script>
</head>
<body>
    <p>Click the following button to see the result: </p>
    <form>
        <input type="button" value="Click Me" onclick="Warn();" />
    </form>
</body>
</html>
```

# Confirmation Dialog Box

```
<html>
<head>
<script type="text/javascript">
  <!--
  function getConfirmation(){
    var retVal = confirm("Do you want to continue ?");
    if( retVal == true ){
      document.write ("User wants to continue!");
      return true;
    }else{
      Document.write ("User does not want to continue!");
      return false;
    }
  }
  //-->
</script>
</head>
```

# Confirmation Dialog Box

```
<body>  
  <p>Click the following button to see the result: </p>  
  <form>  
    <input type="button" value="Click Me" onclick="getConfirmation();" />  
  </form>  
</body>  
</html>
```

# Prompt Dialog Box

```
<html>
<head>
<script type="text/javascript">
    <!--
    function getValue(){
    var retVal = prompt("Enter your name : ", "your name here");
    document.write("You have entered : " + retVal);
    }
    //-->
</script>
</head>
<body>
    <p>Click the following button to see the result: </p>
    <form>
        <input type="button" value="Click Me" onclick="getValue();" />
    </form>
</body>
</html>
```



# Page Printing

Molte volte si desidera inserire un pulsante sulla pagina web per stamparne.

JavaScript consente di implementare questa funzionalità utilizzando la funzione di stampa dell' oggetto window.

La funzione di stampa **window.print()**, stampa la pagina Web corrente quando viene eseguito. È possibile richiamare questa funzione direttamente utilizzando l'evento onclick.

# Page Printing - Esempio

```
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
  <form>
    <input type="button" value="Print" onclick="window.print()" />
  </form>
</body>
```